בס"ד

PPL – HW5

Nofet Damri: 207328865

Elinor Avraham: 209488022

Question 1

1.1:

(a)

We will say that two lazy lists are equal if the following condition are true:

For every n , the element n in the first list = the element n in the second list.

(b)

We need to proof that - $\text{even\_square\_1}[n] = \text{even\_square\_2}[\text{n}]$

Diagnosis: the order of function map, filter is not important because $x\ Mod\ 2 == 0 \Leftrightarrow x^2 Mod 2 == 0$

Let call E_n to the element n in lzl-list.

- After we call even_square_1, For each n, we found the E_n^2 in the lzl-list $\Leftrightarrow\ E_n^2\ Mod 2 == 0$.
- After we call even_square_2, For each n, we found the E_n^2 in the lzl-list $\Leftrightarrow\ E_n\ Mod 2 == 0$.

From the Diagnosis, $x\ Mod\ 2 == 0 \Leftrightarrow x^2 Mod 2 == 0$ i.e:

$E_n^2 \in$ lzl-list after we call even_square_1

$\rightarrow\ E_n^2\ Mod 2 == 0 \rightarrow\ E_n\ Mod 2 == 0$

$\rightarrow E_n^2 \in$ lzl-list after we call even_square_2

In words:

The procedure 'integers-from' returns a lzl-list which contains the natural numbers starts from 0.

The procedure 'even-square-1' first activates 'lzl-filter' and returns the even numbers, and then activates 'lzl-map' and returns the square of

the numbers in the list. That is, we get the square of the even numbers.

The procedure 'even-square-2' first activates 'lzl-map' and returns the square of the numbers in the list, and then activates 'lzl-filter' and returns the even numbers. That is, we get the square of the even numbers.

The return value of these 2 procedures is a lazy list which contain the square of the even numbers. We note that both lists meet the conditions we have defined, that is, the first element in both is the same, and for each n elements taken from each list we will receive identical values.

Question 2:

(a) we will say that 2 procedures are equivalent if the following is true:

For every $x1, x2, ..xn$, and for every succ, fail $(f\$x) = (g(fx))$

(d)

We need to proof that –

$(get\_value\ \$\ a\_list\ key\ succ\ fail) = (g(get\_value\ a\_list\ key\ ))$

Let notice how the get_value$ function works:

First, it looks for value. then,

If it finds the value → it invokes the function succ on the value,

Otherwise → it returns fail.

It is easy to see that the operation is identical to

$(g(get\_value\ a\_list\ key\ ))$,

that because, that the get_value function work like that-

it look for the value, If it finds the value, it return it, Otherwise, it returns fail.

So,

if the value exist- $(g(get\_value\ \ a\_list\ key\ ))$, return the same value like $(get\_value\ \$\ a\_list\ key\ succ\ fail)$, because it invokes the function succ on the value that get_value return.

And if the value is not exist, it return fail exactly like get_value$.

Question 3.1:

(a)

1- sub= {s(s)=s(H) →s=H}

A◦sub = t(s(H),G,H,p,t(E),H)

B◦sub = t(s(H),G,p,p,t(E),k)

2- sub= {G=G}

A◦sub = t(s(H),G,H,p,t(E),H)

B◦sub = t(s(H),G,p,p,t(E),k)

3- sub= {H=p}

A◦sub = t(s(H),G,H,H,t(E),H)

B◦sub = t(s(H),G,H,H,t(E),k)

4- sub= {H=H}

A◦sub = t(s(H),G,H,H,t(E),H)

B◦sub = t(s(H),G,H,H,t(E),k)

5- sub= {t(E)=t(E)}

A◦sub = t(s(H),G,H,H,t(E),H)

B◦sub = t(s(H),G,H,H,t(E),k)

6- sub= {H=k}

A◦sub = t(s(k),G,k,k,t(E),k)

B◦sub = t(s(k),G,k,k,t(E),k)

So we get the assignment: { s=k, k=p, H=k } .

(b)

1- sub= {c=c}

A◦sub = g(c,ν(U),g,G,U,E,ν(M))

B◦sub = g(c,M,g,ν(M),ν(G),g,ν(M))

2- sub= {ν(U)=M}

A◦sub = g(c,ν(U),g,G,U,E,ν(ν(U)))

B◦sub = g(c,ν(U),g,ν(ν(U)),ν(G),g,ν(ν(U)))

3- sub= {g=g}

A◦sub = g(c,ν(U),g,G,U,E,ν(ν(U)))

B◦sub = g(c,ν(U),g,ν(ν(U)),ν(G),g,ν(ν(U)))

4- sub= {G=ν(ν(U))}

A◦sub = g(c,ν(U),g,ν(ν(U)),U,E,ν(ν(U)))

B◦sub = g(c,ν(U),g,ν(ν(U)),ν(ν(ν(U))),g,ν(ν(U)))

5- sub= {U=ν(ν(ν(U)))}

And we get an error, because we activate the 'check occurs' algorithm and we get circular Tying. So, the assignment is not possible.

(c)

1- sub= {ν=ν}

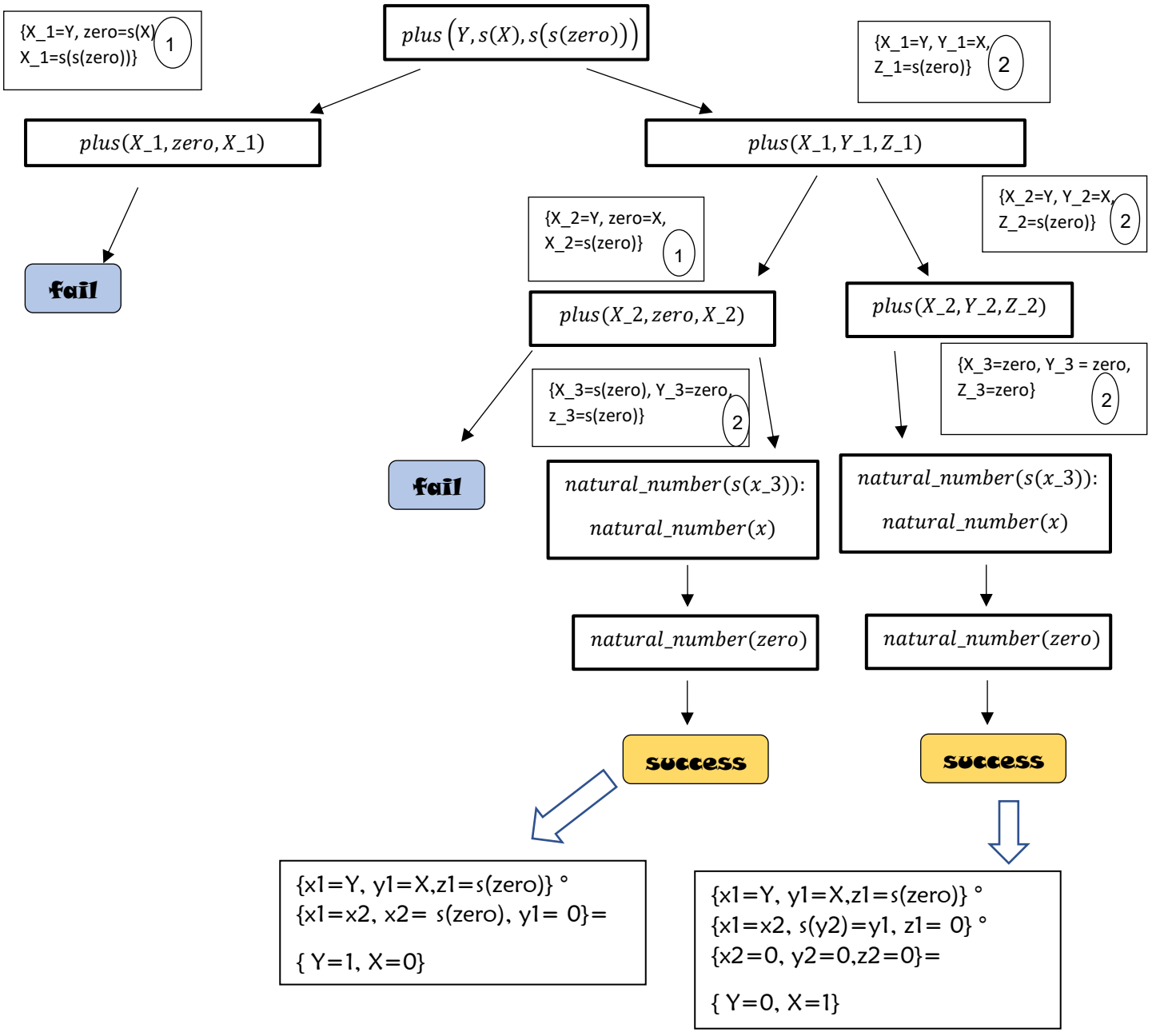
A◦sub = s([ν|[[ν|V]|A]])

B◦sub = s([ν|[ν|A]])]

2- sub= {ν=[ν|V]}

And we get an error, because we activate the 'check occurs' algorithm and we get circular Tying. So, the assignment is not possible.

Question 3.3:



(b) We get 2 satisfy assignment after compose the equations:

1- {Y=1, X=0}

2-{Y=0, X=1}

(c) For a tree to be considered as a success proof tree, it's enough to have one success path. The tree we received above has 2 success paths and is therefore considered a success proof tree.

(d) As we can see this is finite tree, each branch has a point where it gets to its leaf, so by definition there are any infinite paths at this tree and therefore it is a finite tree.